INFORMÁTICA NA EDUCAÇÃO: teoria & prática Porto Alegre, v.21, n.2, mai./ago. 2018. ISSN digital 1982-1654
ISSN impresso 1516-084X

Dekstra: Um ambiente de Aprendizagem Social para Iniciação à Aprendizagem de Programação Usando Esquemas de Concepção e Planos de Programação

Dekstra: a Social Learning Environment for Programming Apprentices Using Design Schemes and Programming Plans

JALVES MENDONÇA NICÁCIO
Instituto Federal de Alagoas

FÁBIO PARAGUAÇU
Universidade Federal de Alagoas

Resumo: A aprendizagem de algoritmos é uma etapa essencial na atividade de programar. No entanto, o aprendiz de programação tem muita dificuldade de elaborar planos de programação. Esse trabalho propõe a concepção e realização de um Ambiente Virtual de Aprendizagem de Programação, baseado no conceito de Ambiente de Aprendizagem Social. No Ambiente proposto, foi disponibilizado na interface, um conjunto de suportes para auxiliar o aprendiz na elaboração dos algoritmos desenvolvidos a partir da biblioteca Blockly. Nesse sentido, foram incorporados no ambiente um conjunto de esquemas de concepção de programas que permite a construção de planos de programação de forma visual e através da interação com um agente inteligente companheiro. A noção de *Scaffolding*, termo utilizado por alguns pesquisadores no contexto da aprendizagem para designar um suporte que é dado ao indivíduo no estágio inicial de aprendizagem, foi fortemente utilizada com vistas a promover a aprendizagem colaborativa dos conceitos iniciais de programação.

Palavras-chave: Ensino de programação. Ambiente de Aprendizagem Social. Andaimento. Plano de Programação. Padrões de Programação.

Abstract: Learning algorithms is an essential step in the programming activity. However, the programming learner finds it very difficult to draw up programming plans. This paper proposes the design and construction of a Virtual Environment Learning Program, based on the concept of Social Learning Environment. In the proposed Environment, a set of supports was provided at the interface to assist the learner in developing the algorithms building with the Blockly library. In this sense, a set of program design schemes was incorporated into the environment that allows the construction of programming plans in a visual way and through the interaction with an intelligent agent. The notion of Scaffolding, a term used by some researchers in the context of learning to designate a support that is given to the individual in the initial stage of learning, was strongly used to promote collaborative learning of early programming concepts.

Keywords: Programming Teaching. Social Learning System. Scaffolding. Programs Plan. Elementary Patterns.

Nicácio, Jalves M.; Paraguaçu, F. Dekstra: Um Modelo de Ambiente de Aprendizagem Social para Aprendizes de Programação Usando Esquemas de Concepção e Planos de Programação. *Informática na Educação*: teoria & prática, Porto Alegre, v. 21, n. 2, p. 120-137, mai./ago. 2018.

1 Introdução

Por repetidas vezes, o processo de ensino e aprendizagem de algoritmos é alvo de pesquisas científicas devido às dificuldades e problemas observados por professores sobre o desempenho dos alunos. Vários estudos (KOLIVER, DORNELES, & CASA, 2004; ALMEIDA et al., 2002; SILVA, CAVALCANTE, & COSTA, 2013) revelam que um número considerável de alunos em períodos iniciais dos cursos da área de computação apresentam dificuldades na aprendizagem dos conceitos fundamentais relacionados à construção de algoritmos.

Diversos problemas no processo de ensino e aprendizagem de algoritmos costumam ser citados: Koliver et al. (2004) explicam que, muitas vezes, o professor inicia a disciplina com uma perspectiva equivocada a respeito das habilidades de seus alunos que, segundo os autores, geralmente chegam na sala de aula sem habilidade de abstração. Almeida et al. (2002) afirmam que o problema é devido a uma forte carga de conceitos abstratos que permeiam todo o conhecimento envolvido na atividade de programação.

O trabalho de Pears et al. (2007) se ocupa em catalogar e classificar diversos estudos sobre o ensino e a aprendizagem de programação, com o objetivo de auxiliar os professores no momento em que estão planejando suas aulas. Conforme o levantamento feito pelos autores em 180 artigos sobre ensino introdutório de programação, as pesquisas se concentram basicamente em quatro categorias: (i) Estudos sobre o currículo de cursos introdutórios de programação; (ii) Métodos pedagógicos que dão suporte ao ensino de programação; (iii) Pesquisas sobre a escolha da linguagem de programação; e (iv) Desenvolvimento de ferramentas computacionais para ensino de programação.

Outra pesquisa que realizou um mapeamento sistemático foi a de Souza, Batista, e Barbosa (2016), que apresentou uma revisão da literatura onde foram coletadas evidências em problemas e dificuldades no ensino e aprendizagem de programação, assim como as diversas soluções propostas para minimizá-los. Após a avaliação de 519 artigos, os autores concluem que há uma tendência de pesquisas em ferramentas de visualização para ajudar os alunos a entenderem os conceitos de programação.

Este trabalho propõe a criação de uma ambiente virtual para aprendizagem de programação baseado no conceito de Ambiente de Aprendizagem Social, como definido em (PARAGUAÇU, 1997). Nesse sentido, partiu-se da hipótese de que programar consiste em interagir com blocos de concepção de programas (chamados ferramentas cognitivas) que são disponibilizados na interface, permitindo a construção interativa de programas.

Com o intuito de desenvolver no aluno as habilidades de resolução de problemas e raciocínio lógico aplicadas à programação, foi proposto neste trabalho uma linguagem de concepção que serve de interface de diálogo entre o aprendiz e o programa a ser criado. Desta forma, esta linguagem permite a elaboração de planos de programação baseados em Padrões Elementares de Programação (WALLINGFORD, 1998).

No ambiente DEKSTRA, proposto neste trabalho, o professor é responsável por cadastrar os problemas que serão resolvidos pelo aluno. Para cada problema cadastrado o professor fornece um modelo mental de resolução, que é um esquema que define de que forma o problema pode ser resolvido. O modelo mental define quais metas devem ser alcançadas para resolução do problema. Cada meta, por sua vez, é realizada a partir dos planos de programação que dizem, passo a passo, como a meta pode ser realizada.

Quando o aluno acessar o ambiente de concepção para realizar as tarefas solicitadas pelo professor, ele vai interagir, na fase de resolução de problemas, com um agente artificial companheiro que foi concebido e implementado, no contexto do ambiente DEKSTRA, para colaborar com o aprendiz na execução da tarefa de programar.

2 Trabalhos Correlatos

Pears et al. (2007) demonstraram que é possível encontrar, na literatura, diversas ferramentas desenvolvidas para o ensino de programação. De fato, com o crescimento da abordagem de ensino a distância, muitos Ambientes Virtuais para Ensino e Aprendizagem (AVEA) são desenvolvidos para o compartilhamento dos mais variados conteúdos.

Em Rocha et al. (2010), os autores propõem o uso da plataforma de ensino a distância MOODLE para promover a aprendizagem de programação utilizando a linguagem de programação PASCAL. Segundo os autores, esta linguagem foi escolhida devido à sua adequação ao ensino inicial de programação imperativa.

Outros artigos também sugerem o uso de linguagens que já são reconhecidas e utilizadas inclusive em ambientes de desenvolvimento profissional. Em Leal (2014), utiliza-se a linguagem de programação C++ numa proposta de ensino da disciplina de programação, onde os alunos devem desenvolver o raciocínio lógico por meio de jogos concretos executados com auxílio de programação. Trabalhos como os de Rocha et al. (2010) e Leal (2014) possuem reconhecida importância no meio acadêmico, no entanto, abordagens de ensino baseadas na utilização de linguagens tais como Pascal e C++, que apresentam sintaxes complexas e tendem a dificultar a escrita dos algoritmos, poderiam se beneficiar com o uso de linguagens de programação visual, tais como SCRATCH ou outras similares.

Outro método de ensino utilizado consiste no uso do pseudocódigo, que é uma forma genérica de escrever um algoritmo, geralmente utilizando uma linguagem mais informal. A função de um pseudocódigo é facilitar o entendimento dos alunos, sem requerer o conhecimento prévio da sintaxe de alguma linguagem de programação. Encontramos, nesta categoria, trabalhos como o de Almeida et al. (2002), que descreve o ambiente denominado AMBAP, e Leite et al. (2013), que avalia o uso da linguagem VisualG em sala de aula.

Para Halstead (2007), a utilização de pseudocódigo pode apresentar a desvantagem da falta de padronização. Segundo o autor, um programador pode ter dificuldade em entender o código-fonte escrito por outra pessoa, dada a sua natureza não estruturada.

Em outra linha, dentro do contexto de aprendizagem colaborativa, existem iniciativas como o trabalho de Adán-Coello et al. (2008), que aborda a dificuldade do professor em dividir os alunos em grupos de trabalho. Segundo os autores, para que se possa garantir a colaboração entre os membros de um grupo, a formação dos grupos não deve ser aleatória e, dessa forma, os autores propõem e desenvolvem duas ferramentas: uma para a avaliação da qualidade dos programas criados pelos alunos e outra para organização dos grupos baseado nos estilos de aprendizagem de cada aluno.

Pode-se perceber que o trabalho desenvolvido por Adán-Coello et al. (2008) está focado em disponibilizar ferramentas que auxiliem o professor no processo de ensino de programação. Entretanto, ferramentas que suportem o processo de aprendizagem não estão no escopo da citada pesquisa.

Por outro lado, os trabalhos de von Wangenheim, Nunes e dos Santos (2014) e Arantes e Ribeiro (2017) estão focados na melhoria do processo de aprendizagem do aluno quando sugerem estratégias para o ensino de conceitos de computação e programação de forma interdisciplinar, usando SCRATCH. A proposta destes autores é contribuir para a formação de um pensamento computacional. A despeito da utilização do programa SCRATCH, tal proposta tem pontos em comum com o presente trabalho, em especial, no tocante ao uso de uma linguagem de programação visual, conforme será descrito neste artigo.

3 Referencial Teórico

As subseções abaixo abordam os principais conceitos utilizados neste trabalho, quais sejam: Ambientes de Aprendizagem Social, Scaffolding, Esquemas de Concepção, Linguagem de Concepção, Planos de Programação e Linguagens de Programação Visual.

3.1 Ambientes de Aprendizagem Social

De acordo com Chan (1991), interações sociais podem influenciar no desenvolvimento cognitivo do indivíduo. Esta afirmação se harmoniza com a hipótese de Vygotsky (2007), que afirma que as interações sociais têm um papel fundamental nas estruturas cognitivas internas. Além disso, a aprendizagem em pares pode aumentar a motivação do aluno, fazendo-o buscar com mais afinco a realização das tarefas propostas (CHAN, 1991). Em um ambiente de aprendizagem social, o aluno pode ser levado a se desdobrar, examinar e defender suas ideias quando desafiado por outro estudante.

Portanto, segundo Chan (1995), Ambiente de Aprendizagem Social (AAS) é um ambiente de aprendizagem interativo suportado por computador, cuja principal característica é a presença de alguns agentes, quer humanos ou artificiais, que interagem assumindo diversos papéis e se envolvem em uma sequência de atividades sociais e educativas. Desta forma, o ambiente de aprendizagem social aceita como pressuposto a teoria histórico-cultural de Vygotsky (2007) sobre Zonas de Desenvolvimento Proximal (ZDP).

O conceito de ZDP, introduzido por Vygotsky (2007), é definido como uma região dinâmica que representa o espaço entre o nível de desenvolvimento real do aluno e o próximo nível (nível de desenvolvimento potencial) que o aluno só poderá alcançar com a assistência de alguém mais experiente. O Nível de Desenvolvimento Real do estudante representa as funções mentais que já amadureceram. Neste nível de desenvolvimento, as atividades podem ser realizadas sem ajuda de outros. No próximo nível de desenvolvimento a ser alcançado, as atividades encontram-se parcialmente desenvolvidas, sendo identificado como o Nível de Desenvolvimento Potencial. Neste nível, o aluno consegue resolver problemas sob a orientação de um adulto, ou em colaboração com outros colegas mais capazes.

Desta forma, um AAS pode ser entendido como um Ambiente Virtual de Ensino e Aprendizagem (AVEA) que possui algumas características que o diferem dos demais, por trabalhar a partir de uma visão colaborativa da aprendizagem. Além disso, segundo Paraguaçu (1997), um AAS é composto pelos seguintes elementos: (i) um conjunto de agentes do sistema, podendo os agentes serem humanos ou artificiais; (ii) um conjunto de ferramentas cognitivas que os agentes podem utilizar; (iii) um conjunto de ferramentas de comunicação, tais como texto, vídeo, etc; (iv) um conjunto de atividades interativas de aprendizagem a partir das quais o aprendiz se envolve com outros agentes.

3.1.1 Ferramentas Cognitivas

Paraguaçu (1999) afirma que as ferramentas cognitivas de um ambiente de aprendizagem social servem como apoio para a colaboração e para a realização de tarefas. Em outras palavras, tais ferramentas são intermediárias da interação entre os agentes.

Ferramentas Cognitivas são, portanto, ferramentas de ajuda à concepção que podem fornecer suporte no processo de aprendizagem dos conceitos que se pretende ensinar. Seguem alguns exemplos de ferramentas que podem auxiliar o aluno na aprendizagem de programação:

(i) Simuladores;

(ii) Calculadoras;

(iii) Espaço de Concepção: é um ambiente virtual para construção de algoritmos durante o processo de aprendizagem de programação.

(iv) Linguagens de Concepção: permite apresentar o conhecimento de maneira estruturada, permitindo que as relações entre as partes do conhecimento sejam explicitadas. Como exemplo de linguagem de concepção, podemos citar mapas conceituais, fóruns de discussão on-line e outros esquemas de concepção como padrões de programação (aplicado ao contexto de aprendizagem de programação).

Em relação ao conjunto de ferramentas cognitivas, o presente trabalho adota um conjunto de esquemas de concepção como *scaffolding* para a aprendizagem de programação. Tal conjunto é

baseado no uso de linguagens de concepção, planos de programação e padrões elementares de programação, conforme é descrito na seção 3.2.

3.2 Esquemas de concepção como *Scaffolding* aplicado à aprendizagem de programação

O termo em língua inglesa *scaffolding* é utilizado na engenharia civil para denotar a colocação de andaimes e outras estruturas de forma a suportar temporariamente os operários de uma construção. Este termo foi utilizado por Wood, Bruner, and Ross (1976) no contexto da aprendizagem, com base nas ideias de Vygotsky sobre a ZDP, para designar um suporte que é dado ao indivíduo no estágio inicial de aprendizagem.

No contexto da educação, o uso de *scaffolding* permite que o aluno consiga realizar uma tarefa que, sem o auxílio de algum suporte, seria muito difícil ou impossível para ele conseguir realizar sozinho. O método então consiste na retirada gradual deste suporte à medida que o aluno começa a compreender a tarefa que deve ser executada.

Segundo os autores, a ideia do uso de *scaffolding* esclarece o que ocorre na ZPD da seguinte forma:

1. A tarefa não é facilitada, mas a quantidade e o tipo de assistência pode ser variável;
2. A responsabilidade é transferida progressivamente, do educador para o aprendiz;
3. O apoio prestado deve ser temporário, retirado gradualmente, conduzindo o aprendiz à autonomia na realização da tarefa.

O conjunto de esquemas de concepção proposto neste trabalho atuará, dentro do ambiente de aprendizagem social, como um *scaffolding* e está baseado na aplicação dos conceitos de Linguagens de Concepção (LC), Planos de Programação e Padrões Elementares de Programação. As próximas seções descreverão sucintamente as definições destes conceitos.

3.2.1 Linguagem de Concepção

De acordo com Paraguaçu (1997), "a linguagem de concepção é utilizada para projetar objetos, especificando o que eles são, o que fazem, sua utilização e sua contribuição à experiência". Em seu trabalho, Paraguaçu procura demonstrar que é possível construir uma LC para melhorar a interação entre o aprendiz de programação e o algoritmo gerado a partir de determinada linguagem de programação, melhorando, desta forma, a compreensão da tarefa de programar.

Em suma, uma LC é capaz de representar um objeto a partir de uma série de elementos e o relacionamento entre esses elementos (WINOGRAD, 1996). Além disso uma LC carrega consigo um conjunto de questões sobre o próprio objeto. É justamente este conjunto de questões que pode ser usado como um *scaffolding*[1], ajudando o aluno a compreender o objeto a partir dos questionamentos feitos sobre os elementos da LC.

Para ilustrar, considere, por exemplo, a operação de atribuição. Esta operação permite ao programador definir o valor que será armazenado em determinada variável. Independente da linguagem de programação utilizada, este comando sempre obedecerá a seguinte regra de montagem:

```
<variavel> <op> <expressão>
```

Onde <op> representa o operador de atribuição, que pode ser simbolizado de várias formas diferentes, dependendo da linguagem de programação. Duas formas típicas de simbolizar este operador são: "=" e ":=". Dentro de determinada linguagem de programação a forma de representar o operador de atribuição é imutável e, desta forma, dizemos que <op> é uma palavra-chave que representa uma ação possível dentro do programa. Por sua vez, <variavel>

[1] Há autores em língua portuguesa, como Bergmann (2013), que já usam o termo andaimento em substituição à palavra em inglês *scaffolding.*

 Porto Alegre, v.21, n.2, mai./ago. 2018. ISSN digital 1982-1654
ISSN impresso 1516-084X

e <expressão> são elementos que podem variar dependendo da situação, e representam argumentos de uma operação de atribuição, dentro do programa.

A partir da representação acima, podemos identificar duas questões que o programador deverá responder para implementar uma atribuição: (1) "A que variável será atribuído um valor?" e (2) "Que valor será atribuído?".

Desta forma, podemos definir Linguagem de Concepção de Programação como o conjunto de palavras-chaves e argumentos que se relacionam para produzir linhas ou blocos de código, podendo-se atribuir a estes argumentos e palavras-chave uma série de questionamentos, conforme demonstrado no exemplo acima.

3.2.2 Planos de Programação

De acordo com Lemos, Lopes, and Barros (2005), planos de programação são abstrações mentais de um programa, que podem ser traduzidos para uma linguagem de programação. Segundo as autoras, programadores experientes conseguem mapear metas de um problema diretamente em planos de programação. Por outro lado, alunos iniciantes possuem apenas habilidade de criar planos naturais (LEMOS, BARROS, & LOPES, 2003). Os planos naturais são diferentes dos planos de programação porque não consideram os aspectos computacionais na solução de um problema.

Por exemplo, um fato comumente conhecido entre professores de programação é que, diferentemente dos alunos iniciantes na primeira linguagem de programação, aqueles que já possuem experiência prévia em alguma linguagem de programação sentem menos dificuldades para aprender a segunda linguagem. Há, portanto, um forte indício de que o conhecimento prévio sobre resolução de problemas computacionais, ou seja, a habilidade de construir planos de programação, seja um pré-requisito para o aprendizado de linguagens de programação.

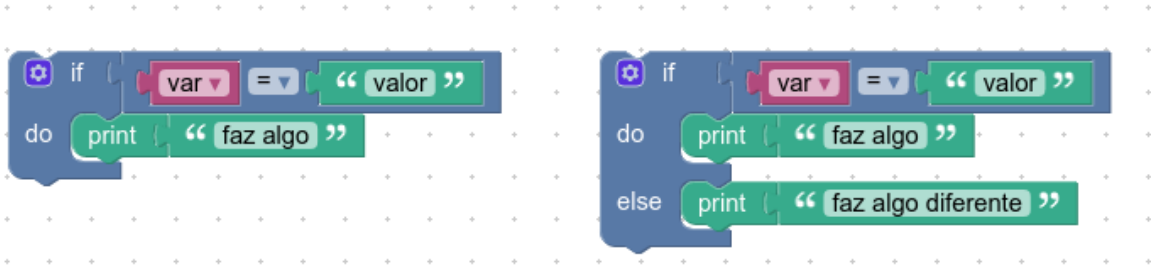
Soloway e outros pesquisadores implementaram vários estudos (SOLOWAY & EHRLICH, 1984; BONAR & SOLOWAY, 1985; JOHNSON & SOLOWAY, 1985) que mostram que programadores experientes possuem a habilidade de resolver problemas partindo da identificação de metas de programação. Após a identificação das metas, os programadores seguem para a criação de planos de programação e, somente depois, transformam-nos em algoritmo executável por computador. Neste sentido, Lemos et al. (2003) identificaram cinco passos que devem ser realizados no processo de resolução de problemas:

1. Identificar os objetivos ou metas do problema;
2. Selecionar planos de programação que alcancem seus objetivos;
3. Selecionar o conjunto de ações que compõe cada plano;
4. Decompor as ações em sub-planos (ações primitivas);
5. Codificar cada sub-plano em linguagem de programação.

Assim, uma vez que o aprendiz precisa desenvolver a habilidade de mapear planos naturais em planos de programação, seria interessante se, de alguma forma, ele pudesse se apoiar no conhecimento prévio de programadores experientes. Pensando nisto, Lemos (2004) propõe um modelo de Processo de Construção de Programas para Aprendizes que considera os planos de programação criados por programadores experientes como elementos-chave para promover a construção de um modelo mental para resolução de problemas.

Como pretendem trazer o conhecimento de programadores experientes para apoiar o desenvolvimento do aprendiz, Lemos et al. (2003) modelam os planos de programação a partir de padrões elementares de programação (The Elementary Patterns Home Page, 2005). Padrões elementares são fragmentos de código que correspondem a certas ações ou operações de alto nível aplicáveis a algum contexto (WALLINGFORD, 1998).

Figura 1: Exemplos de padrões de programação em Linguagem de Programação Visual



Fonte: Autor

A Figura 1 ilustra dois padrões elementares de programação apresentados em Bergin(1999) utilizando linguagem de programação visual. Tais padrões elementares mostram ao aluno como usar simples estruturas de seleção em um programa. Eles foram nomeados pelo autor como *Whether or not* (esquerda da figura) e *Alternative Action* (à direita). Esses dois padrões foram traduzidos neste trabalho respectivamente para "Independente de resposta" e "Ação alternativa".

A Tabela 1 demonstra como representar os mesmos padrões elementares em forma de planos de programação, decompondo-os em ações primitivas e gerando um esquema de concepção.

Tabela 1 – Planos de programação decompostos em ações primitivas

Plano de Programação	Ações Primitivas	Esquema

Independente de Resposta	1) Insere condição
	2) Implementa ação(ões) para condição verdadeira
		`If ($condição){`
		`  $ação;`
		`}`

Ação Alternativa	1) Insere condição
	2) Implementa ação(ões) para condição verdadeira
	3) Implementa ação(ões) para condição falsa
		`If ($condição){`
		`  $ação;`
		`} else {`
		`  $ação;`
		`}`

Fonte: autor.

3.3 Linguagens de Programação Visual

Na literatura, é possível encontrar muitas linguagens de programação desenvolvidas cuja principal finalidade é facilitar o acesso de iniciantes ao conhecimento de programação. É o caso, por exemplo das linguagens: Logo (PAPERT, 1988), Pascal (PACITTI, AKTINSON, & TELES, 1983), Portugol (MANSO, OLIVEIRA, & MARQUES, 2009) e as linguagens de programação visual.

Segundo McIntyre (1998), linguagem de programação visual (VPL) é comumente definida como uma linguagem de programação que utiliza expressões visuais, tais como gráficos, desenhos, animações ou ícones, no processo de programação. Estas expressões visuais podem ser entendidas como interfaces que interpretam graficamente o que pode ser transcrito para

linguagens de programação textuais em ambientes de programação. Assim, as expressões visuais utilizadas formam uma nova sintaxe de programação completamente gráfica.

Linguagens de programação visual possuem vantagens e desvantagens quando comparadas com linguagens de programação textuais. Uma das vantagens é que o uso de representações gráficas de objetos pode ajudar no melhor entendimento sobre programação orientada a objeto. Além disso, as linguagens visuais eliminam detalhes de sintaxe que podem ser um problema para o aluno aprendiz, como o uso de chaves, parêntesis, ponto-e-vírgula, etc.

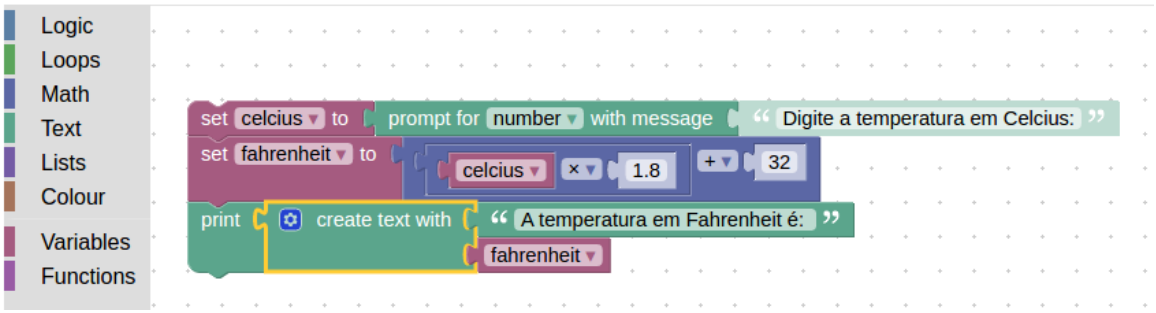
Talvez a melhor vantagem seja o uso que as linguagens de programação visuais podem fazer de metáforas da vida real. Linguagens de programa visuais como Scratch (RESNICK et al.,2009) e Blockly (FRASER et al., 2013), por exemplo, representam pequenos pedaços de código como blocos de montar ou peças de quebra-cabeça. Essas peças podem ser encaixadas umas às outras, tornando explícita a conexão existente entre objetos da linguagem.

Blockly é uma biblioteca para programação desenvolvida pela Google que adiciona um editor de código visual em aplicativos para a plataforma Web e Android. O editor de código do Blockly usa peças que se encaixam, como em um quebra-cabeças, para representar todos os elementos básicos de uma linguagem de programação: ação, sequência, decisão, iteração e estado. (Intro to Computational Thinking: Algorithms - Intro Blockly, 2016).

Blockly é feito de forma que seus blocos se encaixem, mas somente arranjos sintaticamente corretos são permitidos. Esta propriedade liberta o aluno iniciante da preocupação com erros sintáticos, permitindo que ele se concentre na estrutura lógica do código que está tentando criar (FRASER et al., 2013).

O ambiente de programação do Blockly consiste em uma área que contém uma caixa de ferramentas, com blocos previamente definidos, e uma área de trabalho. Qualquer bloco da caixa de ferramentas pode ser selecionado e colocado na área de trabalho. A Figura 2 apresenta o ambiente de programação do Blockly e, na área de trabalho, o código em Blockly que transforma a temperatura medida em celsius para fahrenheit. Note-se que cada um dos blocos tem um entalhe no topo e uma saliência na parte inferior. Esta saliência vai se encaixar no entalhe que fica no topo de outro bloco. Desta forma, os blocos podem ser "empilhados" na sequência em que se quer executar o programa.

Figura 2: Programa feito em Blockly para transformar temperatura medida em Celsius para Fahrenheit.



Fonte: Autor.

4 O Ambiente de Aprendizagem DEKSTRA

Dekstra é um Ambiente Virtual de Aprendizagem baseado no Conceito de Ambiente de Aprendizagem Social. Este ambiente é constituído de um conjunto de esquemas de concepção que oferecem ajuda ao aluno no processo de aprendizagem de programação, usando uma abordagem baseada em resolução de problemas, e permitindo que o aluno trabalhe desde a fase de formulação de uma solução até a visualização da execução de sua solução através de simulador de programa.

 Porto Alegre, v.21, n.2, mai./ago. 2018. ISSN digital 1982-1654
ISSN impresso 1516-084X

O nome escolhido para a ferramenta foi inspirado em um dos mais notáveis cientistas da computação, o professor Edsger W. Dijkstra[2]. Seu artigo "Go To Statement Considered Harmful" (Dijkstra, 1968) influenciou fortemente para a depreciação do comando "Goto" em prol de estruturas de controle, tais como as estruturas de sequência, repetição e seleção. Ele alegava que o artifício era motivo para vários erros de programação.

Trata-se de um ambiente virtual para resolução de problemas de programação capaz de auxiliar o professor a criar uma série de atividades (problemas) para serem realizadas com os alunos. A formulação de um conjunto de questões deve orientar o aluno no processo de concepção do conhecimento, ao ponto que possibilita a reflexão do aluno sobre os aspectos necessários para solucionar tais problemas. Além disso, para a implementação da solução proposta, foi realizada a integração do ambiente com a biblioteca de programação visual Blockly.

O ambiente funciona da seguinte forma: primeiramente o professor deve cadastrar todas as atividades que serão executadas pelos alunos. Além disso, para cada atividade cadastrada, também deve ser cadastrado um modelo mental da solução do problema proposto na atividade. Este modelo mental, posteriormente, será útil para oferecer ajuda ao aluno através de um agente artificial companheiro, um agente reativo capaz de comparar a resposta do aluno com o modelo mental criado pelo professor e, desta forma, disponibilizar a ajuda necessária no momento certo.

Seguindo o que foi descrito na seção 3.1 sobre os Ambientes de Aprendizagem Social, o modelo do ambiente DEKSTRA é composto por: um conjunto de Agentes, Ferramentas Cognitivas, Ferramentas de Comunicação e um conjunto de Atividades Interativas de Aprendizagem.

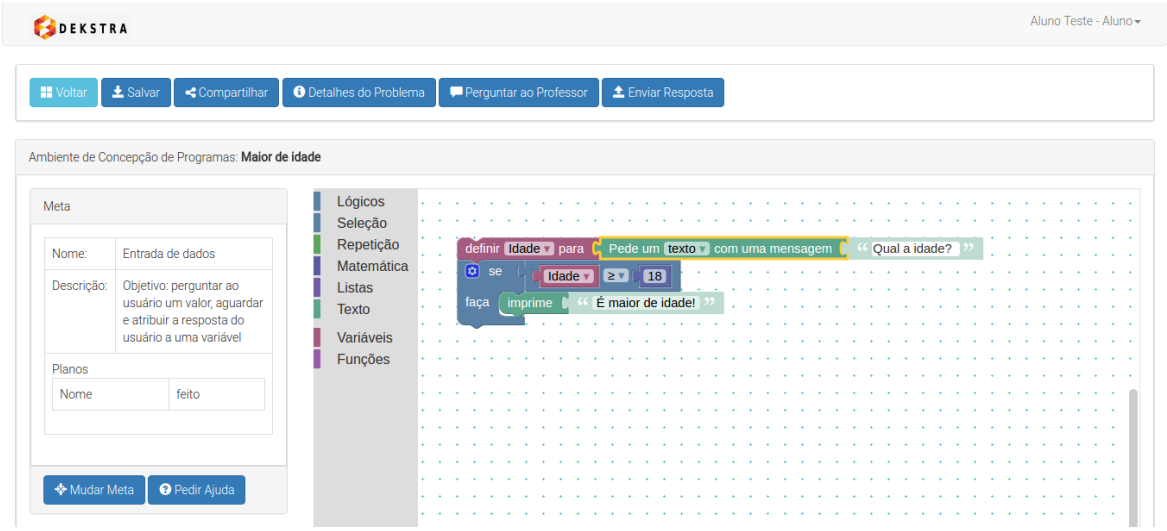
Os agentes do ambiente DEKSTRA podem ser artificiais ou humanos e possuem papéis definidos no sistema. O agente aprendiz (humano) receberá, no ambiente, diversas missões para resolução de problemas de programação. Ele conta com a ajuda do agente companheiro (artificial) que participará sistematicamente da execução das tarefas para o cumprimento dos objetivos de aprendizagem. A função do agente professor (humano) é cadastrar problemas, criando um modelo mental para resolução dos problemas a partir do uso de metas, planos de programação e ações primitivas, além de mediar as ações do agente aluno no ambiente, permitindo que este adquira os conhecimentos necessários para a prática de programação.

As ferramentas cognitivas utilizadas no ambiente DEKSTRA são as seguintes:

- **Ferramentas para realização de tarefas**: o ambiente de concepção, a caixa de ferramentas para programação visual e o simulador de programa;
- **Ferramentas para colaboração**: a solicitação de ajuda ao agente companheiro, a rede de conversação com outros agentes aprendizes;
- **Ferramentas para comunicação**: o ambiente de mediação entre aluno e professor.

[2] O nome DEKSTRA foi formulado a partir da escrita fonética do nome do cientista.

Figura 3: Ambiente de Concepção de Programa



Fonte: Autor.

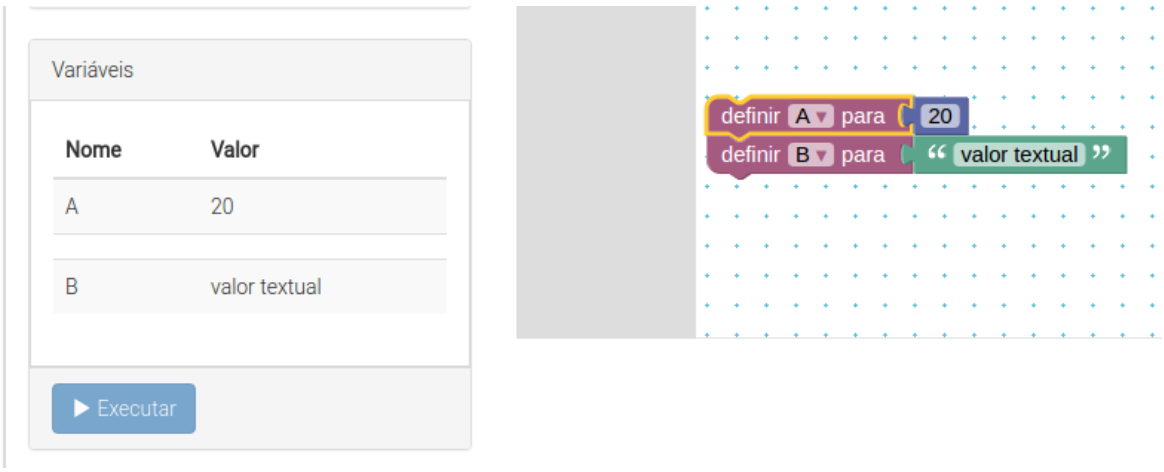
O ambiente de concepção de programas, mostrado na Figura 3, é formado por uma caixa de ferramentas para programação, um simulador de programa e a função de solicitação de ajuda ao agente companheiro. A caixa de ferramentas deve ser utilizada pelo aluno para construção de código-fonte. Tais ferramentas se baseiam no uso de blocos encaixáveis, uma metáfora para programação utilizada no conceito de linguagens visuais de programação, apresentado na seção 3.3.

O simulador de programa (Figura 4) é uma ferramenta integrada ao ambiente de concepção capaz de simular a execução passo-a-passo do programa criado pelo aluno. O simulador permite ao aluno fazer uma avaliação de seu próprio código, acompanhando, a cada passo da execução, as mudanças dos valores das variáveis do programa.

A solicitação de ajuda ao agente companheiro é uma função disponível no ambiente de concepção onde o aluno pode requisitar a ajuda do agente artificial. O agente deverá então avaliar o código-fonte construído pelo aluno até aquele momento e fornecer para o aluno uma questão resgatada da metalinguagem de manipulação existente no modelo mental do problema criado pelo professor. Esta questão deverá permitir que o aluno reflita sobre o que ele deve fazer para resolver uma determinada meta estipulada pelo modelo mental do problema.

A rede de conversação pode ser utilizada pelo aluno para compartilhar com outros alunos tanto os desafios encontrados na resolução de problemas quanto o seu próprio ponto de vista quanto às possíveis respostas a um problema. É um ambiente que permite a interação e troca de conhecimentos entre pares, permitindo que um aluno receba ajuda de outros alunos mais capazes.

Figura 4: Simulador de Programa



Fonte: Autor.

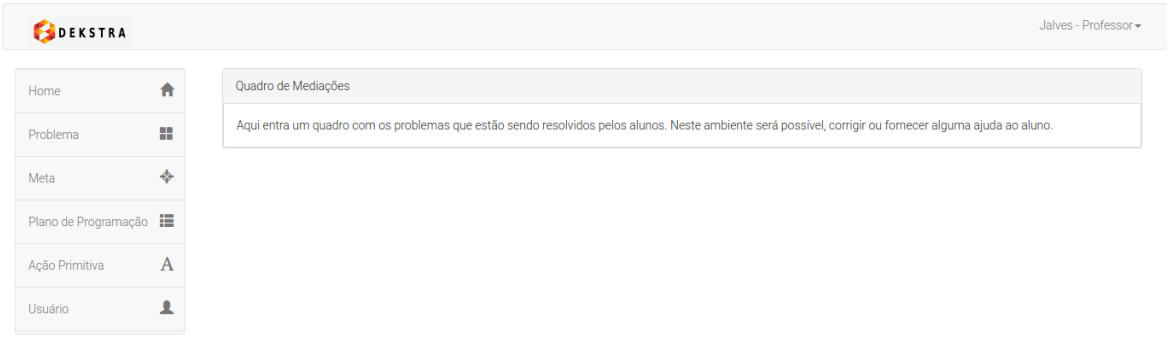
O Ambiente de mediação permite ao aluno solicitar ajuda diretamente ao professor, possibilitando que o professor colabore com a solução que está sendo construída pelo aluno. É também neste ambiente que o aluno submeterá sua resposta ao problema proposto pelo professor para que este possa avaliar a resposta do aluno.

4.1 Visão Geral da Solução Proposta

O ambiente DEKSTRA consiste basicamente de dois subsistemas: Módulo Professor e Módulo Aluno. As telas iniciais dos dois módulos estão ilustradas nas Figuras 5 e 6.

O Módulo do Professor consiste na interface entre o ambiente de aprendizagem e o agente professor. Este subsistema é responsável pelo cadastro dos problemas que serão resolvidos pelos alunos, bem como o cadastro do modelo mental de implementação de cada problema. O modelo mental é composto pelo cadastro de todas as metas, planos e ações necessários para orientação do aluno na resolução do problema. Além disso, o Módulo do Professor também é responsável pelo cadastro dos usuários (alunos e professores) que utilizarão o sistema.

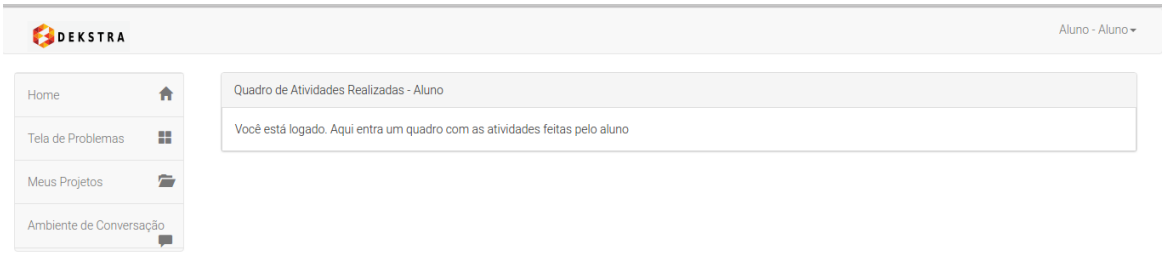
Figura 5: Tela inicial para o módulo do Professor



Fonte: Autor

O Módulo Aluno permite a comunicação entre o aluno e o ambiente de aprendizagem. Este módulo possibilita que o aprendiz visualize os problemas cadastrados previamente pelo professor para que, em seguida, ele possa fornecer uma solução implementada em linguagem visual de programação.

Figura 6: Tela inicial para o módulo Aluno.



Fonte: Autor.

4.2 Projeto Arquitetural

O ambiente DEKSTRA é uma aplicação Web que se baseia na arquitetura cliente-servidor, na qual cada instância de um cliente envia requisições de dados para o servidor conectado e espera pela resposta. O servidor, por sua vez, recebe tal requisição, processa a informação e devolve o resultado para o cliente.

Para implementar essa arquitetura foi utilizado o framework PHP Laravel (OTWELL, 2016), que facilita a implementação de aplicações Web seguindo essa estrutura arquitetural.

Dentro do contexto da arquitetura cliente-servidor, o lado cliente do sistema é dividido em duas visões de acesso: a visão do aluno e a visão do professor. O lado servidor é constituído por diversos módulos de cadastro desenvolvidos neste trabalho. Além disso, foram utilizados dois frameworks de terceiros e as bases de dados. Na Figura 7, está representada a arquitetura do sistema.

De acordo com o projeto arquitetural (Figura 7), no lado cliente do ambiente DEKSTRA, a visão do aluno é representada por uma interface Web que lhe permite visualizar o problema a ser resolvido, consultar os problemas que já foram resolvidos, solicitar ajuda do agente companheiro, solucionar o problema, executá-lo e submeter a solução para o professor. Além disso, a visão do aluno também permite o compartilhamento da solução do aluno e a visualização dos comentários emitidos por ele mesmo ou pelos colegas.

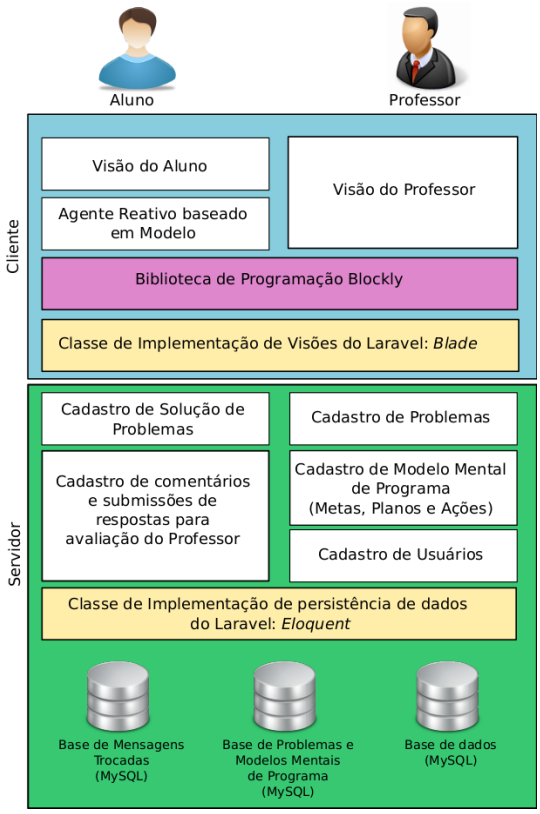
Também é na visão do aluno que o agente inteligente companheiro encontra-se situado. Ele reage às ações do aluno no ambiente de concepção, percebendo as alterações que este provoca no ambiente e respondendo com ações adequadas à situação. Uma descrição mais detalhada sobre o agente inteligente companheiro encontra-se na seção 4.3.

No que lhe concerne, a visão do professor permite que este agente mantenha as informações necessárias para o funcionamento do ambiente – problemas e modelos mentais de programas. Além disso, o professor poderá analisar as soluções submetidas pelos alunos e emitir comentários que podem ser visualizados pelos alunos.

Do lado servidor, os módulos de cadastro gerenciam a manutenção dos problemas, modelos mentais de programação, dos comentários e dos usuários. Neste, o professor pode inserir, remover e editar cada uma das informações citadas.

As implementações pertencentes ao framework Laravel encontram-se no lado servidor e são utilizadas para facilitar a construção das telas de interfaces com o usuário. Para a persistência de dados, foi utilizada a tecnologia Eloquent, parte integrante do framework Laravel, a qual é responsável pelo mapeamento objeto-relacional dos dados, possibilitando a implementação de um sistema orientado a objetos e armazenar os dados em um banco relacional.

Figura 7: Projeto arquitetural do ambiente DEKSTRA



Fonte: Autor.

4.3 Implementação do Agente Inteligente Companheiro

O agente companheiro implementado no ambiente DEKSTRA foi desenvolvido de acordo com o conceito de agente reativo simples. Seu objetivo é auxiliar o aluno no desenvolvimento de programas dentro do ambiente de concepção do DEKSTRA.

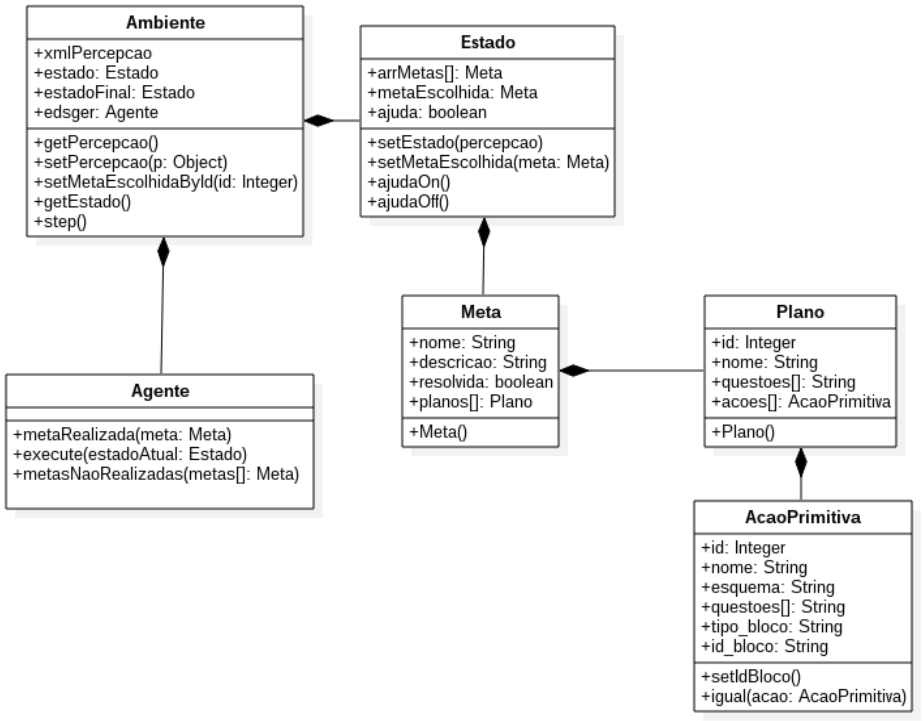
A Figura 8 apresenta o diagrama de classes dos componentes que integram o sistema do agente companheiro. Nesta figura, está representada toda estrutura e relações das classes que servem de modelo para os objetos do sistema do agente companheiro.

A classe Ambiente guarda o estado atual do ambiente de concepção. O estado atual é obtido através da percepção das ações do aluno. A codificação gerada pelo aluno é capturada do ambiente de concepção e repassada ao estado a partir da propriedade xmlPercepção (passado em formato XML).

As classes Meta, Plano e AcaoPrimitiva definem um estado do sistema. De acordo com o modelo do problema, um estado é formado por muitas metas, uma meta é formada por vários planos e um plano é formado por várias ações primitivas.

A classe Agente define os métodos ou ações do agente. Um agente deve verificar se uma meta já foi alcançada, deve listar as metas que ainda não foram cumpridas e deve executar o programa do agente que contém as regras que o ajudam a decidir que ação será executada.

Figura 8: Diagrama de classe do agente reativo baseado em modelo

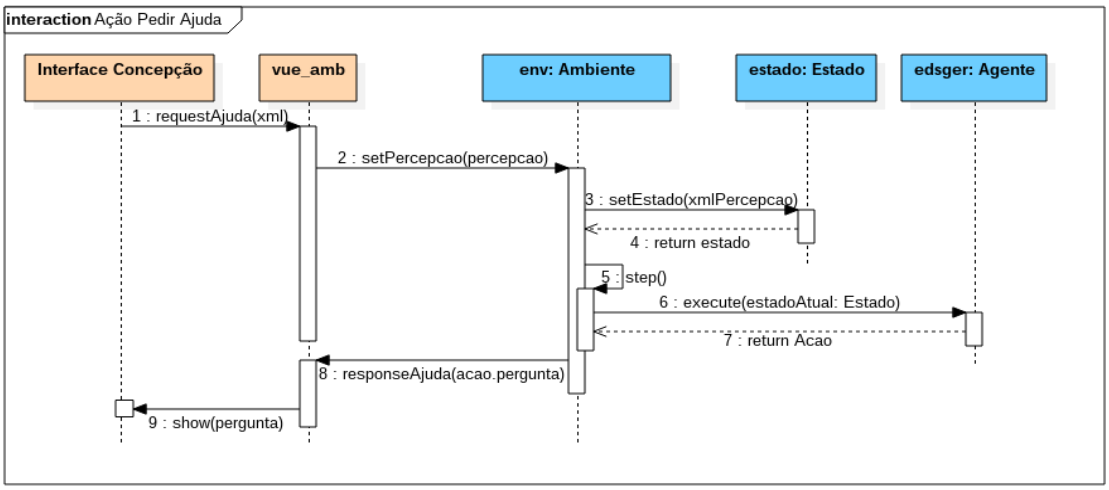


Fonte: Autor.

4.3.1 Diagrama de atividade: Ação pedir ajuda

A decisão mais complexa para o agente é quando o usuário requisita sua ajuda para continuar resolvendo um determinado problema. Neste caso, o agente deve disponibilizar para o aluno uma questão pertinente ao problema que o mesmo está resolvendo. Esta questão disponibilizada pelo agente é retirada do próprio modelo do problema, criado pelo professor. A Figura 9 apresenta um fluxo de atividades que ocorre quando um aluno pede ajuda ao agente companheiro.

Figura 9: Diagrama de atividade do agente reativo baseado em modelo - ação: Pedir Ajuda.



Fonte: Autor.

De acordo com a Figura 9, inicialmente o usuário solicita ajuda do agente através de um botão existente na interface do ambiente de concepção, gerando uma requisição de ajuda (1) que é enviada para um objeto de controle reativo da interface, denominado de vue_amb. O objeto de controle vue_amb foi implementado utilizando o framework vue.js.

A requisição produzida pela interface entrega ao objeto vue_amb uma cópia do esquema criado pelo aluno até o momento da solicitação de ajuda. De posse deste esquema, o objeto de controle envia para o ambiente do agente uma percepção do estado atual do programa do aluno (2).

O estado do ambiente é atualizado (3 e 4) e o ambiente do agente companheiro pode finalmente executar o método step() (5), responsável por realizar uma consulta ao agente (6). Em seguida, o agente companheiro usará seu programa de agente para decidir que ação é mais adequada (7).

Percebendo que o usuário solicitou sua ajuda para solucionar o problema proposto pelo professor, o agente então retornará uma ação do tipo "perguntar <questão>"(7), enviando assim uma das questões que foram previamente cadastradas pelo professor. Portanto, é responsabilidade do programa do agente decidir que questão é mais adequada para ser exibida ao aluno em determinado momento.

5 Considerações finais

Este artigo apresentou o desenvolvimento de um Ambiente de Aprendizagem Social para auxílio ao aprendizado de princípios de programação. Este ambiente é constituído de um conjunto de esquemas de concepção que oferecem ajuda ao aluno no processo de aprendizagem de programação. A abordagem de ensino baseada em resolução de problemas permite que o aluno trabalhe desde a fase de formulação de uma solução até a visualização da execução de seu código-fonte através de simulador de programa.

Inicialmente, um protótipo do ambiente foi construído no contexto de uma pesquisa de mestrado. Após observadas as melhorias que poderiam ser aplicadas, iniciou-se a implementação de uma versão completa do ambiente. No momento em que este artigo foi escrito, o DEKSTRA já estava modelado e implementado, restando executar ainda alguns procedimentos técnicos para disponibilização do ambiente na Web. Dando prosseguimento a esta pesquisa, é intenção dos autores realizar alguns experimentos com o ambiente DEKSTRA em sala de aula.

Como o ambiente DEKSTRA baseia-se na teoria sócio-interacionista proposta por Vygotsky (2007), o sistema estimula as interações sociais tanto entre agentes humanos quanto entre humanos e agentes artificiais. O conhecimento dos alunos vai se desenvolvendo a partir destas interações.

O ambiente DEKSTRA foi pensado como uma ferramenta para ser utilizada em aulas práticas de programação em laboratório, podendo também ser utilizado em modalidade de ensino a distância.

O conjunto de atividades disponibilizado pelo professor a partir da ferramenta, e o modelo mental dos programas podem auxiliar o aluno a compreender as ações necessárias para criar um algoritmo.

Acreditamos que o aprendiz iniciante em programação pode obter melhores resultados se estiver inserido em um ambiente onde possa obter ajuda a partir dos esquemas de concepção disponibilizados e a partir da interação com os colegas e o professor.

Com o uso do ambiente DEKSTRA, espera-se que o aluno aprenda a criar modelos mentais de programas, independente da linguagem de programação que ele precise utilizar.

Reconhecimento

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Referências

ADÁN-COELLO, J., DE MENEZES, W., DE FARIA, E., & TOBAR, C. (2008). Conflito sócio-cognitivo e estilos de aprendizagem na formação de grupos para o aprendizado colaborativo de programação de computadores. Brazilian Journal of Computers in Education, 16(03).

ALMEIDA, E. S. D., COSTA, E. D. B., SILVA, K. D. S., PAES, R. D. B., ALMEIDA, A. A. M., & BRAGA, J. D. H. (2002). Ambap: Um ambiente de apoio ao aprendizado de programação. X Workshop sobre Educação em Informática.

ARANTES, FLÁVIA LINHALIS; RIBEIRO, PAULA EDUARDA JUSTINO. Desenvolvimento do Pensamento Computacional com Valores da Ética Hacker. Informática na Educação: teoria & prática, Porto Alegre, v. 20, n. 2, p. 188-206, mai./ago. 2017.

ASTRACHAN, O. & WALLINGFORD, E. (1998), 'Loop Patterns', *Patterns Languages of Programs*.

BERGMANN, G. L. Andaimento: estratégia de aprendizado vivenciada em aulas de línguas estrangeiras. Cadernos de aplicação: pesquisa e reflexão em educação básica, v. 26, n. 2, p.83-86, 2013.

BERGIN, J. (1999), 'Patterns for selection version 4', Disponível em <http://csis.pace.edu/ bergin/patterns/Patternsv4.html>

BONAR, J. & SOLOWAY, E. (1985), 'Preprogramming Knowledge: A Major Source of Misconceptions in Novice Programmers.', *Human-Computer Interaction* **1**(2), 133-161. Disponível em: <http://dblp.uni-trier.de/db/journals/hhci/hhci1.html>

CHAN, T. W. (1991), Integration-Kid: A Learning Companion System, *in* International Joint Conferences on Artificial Intelligence Organization, ed., .

CHAN, T.-W. (1995), 'Artificial Agents in Distance Learning', *International Journal of Educational Telecommunications* **1**(2), 263–282. Disposnível em: <https://www.learntechlib.org/p/15163>

DIJKSTRA, E. W. (1968), 'Go To statement considered harmful', *Comm. ACM* **11**(3), 147-148.

FRASER, N. & OTHERS (2013), 'Blockly: A visual programming editor', *URL: https://developers.google.com/blockly/guides/overview*.

HALSTEAD, N. (2007), 'Uses of Pseudo Code in Development'. Disponível em: <http://blog.assemble-ron.com/2007/06/03/uses-of-pseudo-code-in-develop>. Acesso em: 4 de Agosto de 2017.

JOHNSON, W. L. & SOLOWAY, E. (1985), PROUST: Knowledge-based program understanding, *in* 'IEEE, Transactions on Software Engineering', pp. 369-380.

KOLIVER, C.; DORNELES, R. V. & Casa, M. E. (2004), Das (Muitas) Dúvidas e (Poucas) Certezas do Ensino de Algoritmos, *in* 'Anais do XXIV Congresso da Sociedade Brasileira de Computação. p. 949-960.'.

LEAL, A. V. A. (2014), 'Ensino de Programação no Ensino Médio Integrado: Uma Abordagem Utilizando Padrões e Jogos com Materiais Concretos.', Master's thesis, Universidade Federal de Goiás.

LEITE, V. M.; SENEFONTE, H. C. M.; BARBOSA, C. R. & SEABRA, R. D. (2013), 'VisuAlg: Estudo de Caso e Análise de Compilador destinado ao ensino de Programação''Nuevas Ideas en Informática Educativa TISE 2013'.

LEMOS, M. A. (2004), 'Uma Abordagem Baseada em Padrões Elementares para Aprendizado de Programação', PhD thesis, Universidade de São Paulo.

LEMOS, M. A.; BARROS, L. N. & LOPES, R. D. (2003), Uma Biblioteca Cognitiva para o aprendizado de programação, *in* 'CONGRESSO NACIONAL DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO'.

LEMOS, M. A.; LOPES, R. D. & BARROS, L. N. (2005), Avaliação do ensino-aprendizagem de programação usando uma abordagem baseada em padrões elementares de programação, *in* 'XXXIII Congresso Brasileiro de Ensino de Engenharia'.

MANSO, A.; OLIVEIRA, L. & MARQUES, CÉ. G. (2009), 'Portugol IDE–Uma ferramenta para o ensino de programação', PAEE.

MCINTYRE, D. (1998), 'Comp.Lang.Visual - Frequently-Asked Questions List', .

OTWELL, T. (2016), 'Laravel Documentation', *Laravel.[Online]*.

PACITTI, T.; AKTINSON, C. P. & TELES, A. A. D. S.LTC, ED. (1983), *Programação e Métodos Computacionais*, Rio de Janeiro.

PAPERT, S. (1988), *Logo: computadores e educação.*, Brasiliense, São Paulo.

PARAGUACU, F. (1997), 'VYGOTSKY: un environnement d'apprentissage social pour la programmation fondé sur la collaboration entre agents d'aide à la conception par cas', PhD thesis, Aix-Marseille 3

PARAGUAÇU, F. (1999). Fazendo a comunicação entre agentes artificiais colaborativos alfabetizadores e agentes humanos efetiva. In: MOURA, D. (Ed.). Os múltiplos usos da língua. Maceió: Edufal, 1999. p. 150–152.

PEARS, A.; SEIDMAN, S.; MALMI, L.; MANNILA, L.; ADAMS, E.; BENNEDSEN, J.; DEVLIN, M. & PATERSON, J. (2007), A survey of literature on the teaching of introductory programming, *in* 'Working group reports on ITiCSE on Innovation and technology in computer science education', ACM, New York, NY, USA, pp. 204--223.

RESNICK, M.; MALONEY, J.; MONROY-HERNÁNDEZ, A.; RUSK, N.; EASTMOND, E.; BRENNAN, K.; MILLNER, A.; ROSENBAUM, E.; SILVER, J.; SILVERMAN, B. & KAFAI, Y. (2009), 'Scratch: Programming for All', *Commun. ACM* **52**(11), 60--67.

ROCHA, P. S.; FERREIRA, B.; MONTEIRO, D.; NUNES, D. D. S. C. & DO NASCIMENTO GÓES, H. C. (2010), 'Ensino e Aprendizagem de Programação: Análise da aplicação de proposta metodológica baseada no Sistema Personalizado de Ensino', *RENOTE* **8**(3).

SILVA, M. T.; CAVALCANTE, M. C. T. C. & COSTA, E. B. (2013), Explorando correlações em Programação: um estudo focado no processo seletivo e em disciplinas correlatas, *in* 'Anais do XXIV Simpósio Brasileiro de Informática na Educação'.

SOLOWAY, E.; EHRLICH, K. (1984) Empirical studies of programming knowledge. IEEE Trans. Software Engineering, IEEE Computer Society Press, SE-10, n. 5, p. 595–609, September 1984.

SOUZA, D.; BATISTA, M.; BARBOSA, E. (2016). Problems and Weaknesses in the Teaching and Learning of Programming: A Mapping Review. Brazilian Journal of Computers in Education 24. 39

VYGOTSKY, L. S. (2007), *A formação social da mente*, Martins Fontes, São Paulo.

WALLINGFORD, E. (1998), Elementary Patterns and their Role in Instruction., *in* 'OOPSLA'98 Educators Symposium Notes'.

VON WANGENHEIM, C. G.; NUNES, V. R. & DOS SANTOS, G. (2014), 'Teaching Computing with SCRATCH in Elementary Schools – A Case Study', *Brazilian Journal of Computers in Education* **22**(03), 115.

WINOGRAD, T. (1996), *Bringing Design to Software*, ACM Press.

WOOD, D. J.; BRUNER, J. S. & ROSS, G. (1976), 'The Role of Tutoring in Problem Solving', *Journal of Child Psychiatry and Psychology* **17**(2), 89--100.

(2016), 'Blockly Games', Google, Acesso em: 07/10/2016.

(2016), 'Maze', Google, Acesso em: 07/10/2016.

(2016), 'Intro to Computational Thinking: Algorithms - Intro Blockly', Virginia Polytechnic Institute and State University, Acesso em: 07/10/2016.

(2005), 'The Elementary Patterns Home Page', University of Northern Iowa, Iowa, homepage de padrões elementares, integra a comunidade de pesquisadores, trabalhos realizados e em andamento na área. Acesso em: 14 out. 2016.

(1998), 'ChiliPLoP: Southwestern Conference on Pattern Languages of Programs', The Hillside Group, Acesso em 21 out 2016.

Recebido em outubro de 2017
Aprovado para publicação em julho de 2018

Jalves Mendonça Nicácio
Instituto Federal de Alagoas, IFAL, Brasil, jalves.nicacio@ifal.edu.br

Fábio Paraguaçu Duarte da Costa
Programa de Pós-Graduação em Modelagem Computational de Conhecimento – Universidade Federal de Alagoas – UFAL, Brasil, fabioparagua2000@gmail.com